(Preprint) AAS 17-832

LOW-THRUST TRAJECTORY OPTIMIZATION WITH SIMPLIFIED SQP ALGORITHM

Nathan L. Parrish,[*] and Daniel J. Scheeres[†]

The problem of low-thrust trajectory optimization in highly perturbed dynamics is a stressing case for many optimization tools. Highly nonlinear dynamics and continuous thrust are each, separately, non-trivial problems in the field of optimal control, and when combined, the problem is even more difficult. This paper describes a fast, robust method to design a trajectory in the CRTBP (circular restricted three body problem), beginning with no or very little knowledge of the system. The approach is inspired by the SQP (sequential quadratic programming) algorithm, in which a general nonlinear programming problem is solved via a sequence of quadratic problems. A few key simplifications make the algorithm presented fast and robust to initial guess: a quadratic cost function, neglecting the line search step when the solution is known to be far away, judicious use of endpoint constraints, and mesh refinement on multiple shooting with fixed-step integration.

INTRODUCTION

Low-thrust, electric propulsion systems compare favorably to chemical propulsion systems because the specific impulse (Isp) is generally an order of magnitude higher for electric propulsion. However, the use of low-thrust propulsion complicates mission design. While an impulsive trajectory can be fully represented with a finite, small number of variables, continuous-thrust trajectories are in principal of infinite dimensions. Thus, we must be clever to reduce the size of the continuous-thrust problem while preserving its accuracy.

Most nonlinear optimization methods require an initial guess that is "close" to an optimal solution, or at least "close" to a feasible solution. However, developing a good initial guess can be as difficult a problem as optimizing in the first place. In general, to find a good initial guess, we must make some assumption about the structure of the solution. Since there are many local optima to these problems, which one we arrive at is a function of the initial guess. The approach demonstrated in this paper is capable of finding optimal trajectories even when no initial guess is available, so it is not necessary to assume any particular structure for the solution.

Another challenge that this paper overcomes is that general nonlinear programming (NLP) solvers can be very slow for specific problems. Some of the most widely-used NLP solvers include SNOPT[1], IPOPT[2], MATLAB fmincon, and KNITRO. Some examples of these NLP solvers in trajectory optimization include: NASA Goddard's GMAT (General Mission Analysis Tool) and its

[*] PhD Candidate, Colorado Center for Astrodynamics Research, University of Colorado Boulder, ECNT 431, UCB, Boulder, CO 80309.
[†] Distinguished Professor, A. Richard Seebass Chair, Colorado Center for Astrodynamics Research, University of Colorado Boulder, ECNT 431, UCB, Boulder, CO 80309.

CSALT collocation tool; PSOPT (PseudoSpectral OPTimization) open-source collocation tool[3–5]; NASA Goddard's EMTG (Evolutionary Mission Trajectory Generator)[6]; NASA Johnson's Copernicus Trajectory Design and Optimization System; and NASA JPL's MALTO (Mission Analysis Low Thrust Optimizer)[7].

While these solvers are powerful and adept at solving a wide array of problems, the authors of the present work found that they can be very slow for low-thrust trajectory optimization – particularly as the problem size grows or as the linear assumptions internal to the optimizer break down. With the exception of IPOPT (which is open-source), high-performance NLP solvers are generally proprietary and each is set up as an "engineering black box". When the algorithm encounters difficulties with an optimization problem, there is very little the user can do to improve the situation. The present research implements a simplified version of the SQP (sequential quadratic programming) algorithm in such a way that trajectory optimization problems are solved in a small fraction of the total number of iterations and thus, a small fraction of the total computation time.

THE CIRCULAR RESTRICTED 3 BODY PROBLEM

The circular restricted three body problem (CRTBP) model assumes that the spacecraft is massless in comparison to the two primaries (here, the Earth and Moon). The primaries orbit their barycenter in a circular orbit.

Dimensionless units are used so that all state variables are on the order of 1. One distance unit (DU) is equal to the mean distance between Earth and Moon, or 384747.962856037 km. The non-dimensional mass ratio μ is equal to 0.012150585609624. A synodic reference frame is used such that the Earth is fixed at the point $[-\mu, 0, 0]^T$, and the Moon is fixed at the point $[1-\mu, 0, 0]^T$. This reference frame is illustrated in Fig. 1.

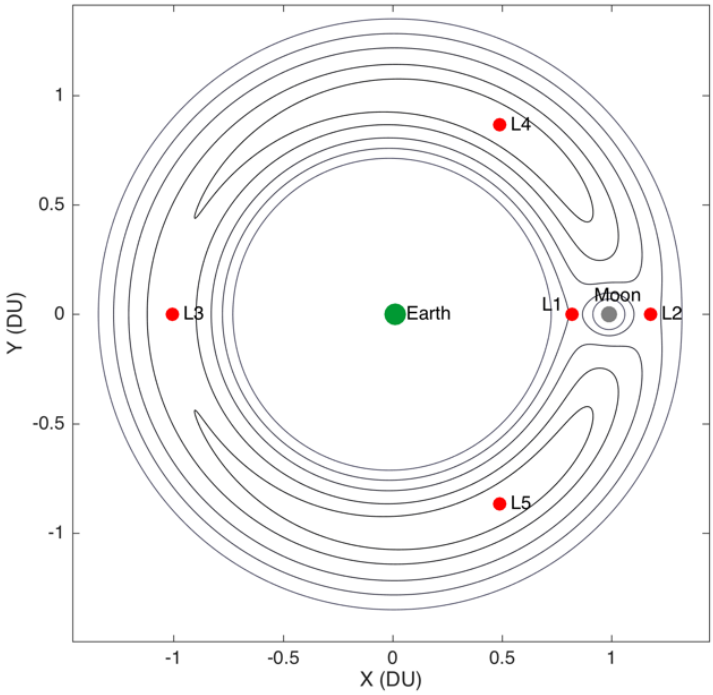


Fig. 1. The Earth-Moon synodic reference frame, with libration points L_1 through L_5 labeled. Note that Earth and Moon are both fixed along the x-axis at $(-\mu)$ and $(1-\mu)$, respectively.

The equations of motion for the system in this rotating frame are given by

$$\ddot{x} = -\left(\frac{(1-\mu)}{r_1^3}(x+\mu) + \frac{\mu}{r_2^3}(x-1+\mu)\right) + 2\dot{y} + x + a_{u,x} \tag{1}$$

$$\ddot{y} = -\left(\frac{(1-\mu)}{r_1^3}y + \frac{\mu}{r_2^3}y\right) - 2\dot{x} + y + a_{u,y} \tag{2}$$

$$\ddot{z} = -\left(\frac{(1-\mu)}{r_1^3}z + \frac{\mu}{r_2^3}z\right) + a_{u,z}. \tag{3}$$

The use of simplified dynamical models here is justified because the algorithm's effectiveness would not be affected by higher-fidelity perturbations.

THE OPTIMAL CONTROL PROBLEM

We find it helpful to write the optimal control problem in different ways. Ultimately, the problem we want to solve is:

$$\text{minimize: } f_{path} = \int \|\vec{u}(t)\|^p \, dt \tag{4}$$

$$\begin{aligned} \text{subject to: } & \vec{h}(\vec{x},\vec{u},t) = \vec{0} \\ & \vec{g}(\vec{x},\vec{u},t) \leq \vec{0} \end{aligned} \tag{5}$$

When the exponent p in the objective function is equal to 1, we are finding the minimum fuel solution. When $p = 2$, we are finding what is commonly referred to as the "minimum energy" solution. The radius of convergence is much larger when $p = 2$, so we use that cost function initially. In this paper, we will leave $p = 2$, but the present authors and others have shown previously how homotopy can be used to transition to the minimum fuel solution[8–11].

In this work, the equality constraints $\vec{h}(\vec{x},\vec{u},t)$ are used to enforce the problem dynamics and to constrain the endpoints. The inequality constraints $\vec{g}(\vec{x},\vec{u},t)$ are used to limit the thrust magnitude. We define the Lagrangian functional associated with the NLP as

$$\mathcal{L}(\vec{x},\vec{u},t,\vec{\lambda},\vec{\mu}) = f(\vec{x},\vec{u},t) + \vec{\lambda}\cdot\vec{h}(\vec{x},\vec{u},t) + \vec{\mu}\cdot\vec{g}(\vec{x},\vec{u},t). \tag{6}$$

Along with the KKT conditions constraining the Lagrange multipliers $\vec{\lambda}$ and $\vec{\mu}$, the NLP problem is now rewritten in the following form.

$$\text{minimize: } \mathcal{L}(\vec{x},\vec{u},t,\vec{\lambda},\vec{\mu}) \tag{7}$$

$$\begin{aligned} \text{subject to: } & \vec{h}(\vec{x},\vec{u},t) = \vec{0} \\ & \vec{g}(\vec{x},\vec{u},t) \leq \vec{0} \end{aligned} \tag{8}$$

where $\mathcal{L}$, $\vec{h}$, and $\vec{g}$ are all nonlinear functions.

SEQUENTIAL QUADRATIC PROGRAMMING

Since it is, in general, hard or impossible to analytically solve a nonlinear problem, we need to convert the problem into some form that is possible to solve. The Sequential Quadratic Programming (SQP) algorithm is a robust option. The basic concept is to solve a series of quadratic sub-

problems, each of which approximate the nonlinear problem at the current iteration[1,12]. The quadratic problem at each problem is constructed as follows:

$$\text{minimize: } \nabla\mathcal{L}\left(\vec{x}_k, \vec{u}_k, t, \vec{\lambda}_k, \vec{\mu}_k\right) \cdot \delta\vec{X} + \frac{1}{2}\delta\vec{X} \cdot H\mathcal{L}\left(\vec{x}_k, \vec{u}_k, t, \vec{\lambda}_k, \vec{\mu}_k\right) \cdot \delta\vec{X} \tag{9}$$

$$\begin{aligned} \text{subject to: } & \vec{h}(\vec{x}_k, \vec{u}_k, t) + \nabla\vec{h}(\vec{x}_k, \vec{u}_k, t) \cdot \delta\vec{X} = \vec{0} \\ & \vec{g}(\vec{x}_k, \vec{u}_k, t) + \nabla\vec{g}(\vec{x}_k, \vec{u}_k, t) \cdot \delta\vec{X} \leq \vec{0} \end{aligned} \tag{10}$$

where $\vec{\square}_k$ are the variables about which the current iteration is linearized and $\delta\vec{X}$ is the update to all of the optimization variables. After solving the quadratic problem at iteration k, the optimization variables are updated as

$$\vec{X}_{k+1} = \vec{X}_k + \alpha \cdot \delta\vec{X} \tag{11}$$

where α is a scalar in the range $(0, 1]$ and is determined by performing a line search. Here, the line search is implemented by testing a range of α and choosing the value that minimizes the defect constraint violations. It was found that when solving with a poor initial guess, it is best to leave $\alpha = 1$ for the first several iterations. Then, when the solution is "near" converging, perform the line search at every iteration.

SECOND-ORDER CONIC PROGRAMMING

Second-Order Conic Programming (SOCP) is a well-defined class of optimization problems for which many well-established, high-performance solvers exist. If the optimal control problem can be approximated as a SOCP, then we can solve the approximate problem quickly and efficiently. An SOCP problem consists of some set of the following: linear equality constraints; quadratic inequality constraints; linear objective; and quadratic objective. SOCP and QP (quadratic programming) are very similar, with the distinction that QP only allows inequality constraints to be up to linear, while SOCP allows inequality constraints to be up to quadratic. In the present implementation, quadratic inequality constraints were used to limit the square of the thrust magnitude (which is identical to limiting the thrust magnitude).

The JuMP open-source modeling language[13] was used in the Julia programming language[14]. A great advantage of the JuMP modeling language is that it provides a common, user-friendly interface for a variety of solvers. Two solvers were used in this work: IPOPT (Interior Point OPTimizer), which is open-source, and Gurobi, which is free for academia. It was found that for these problems, Gurobi is some 20-40X faster, making it the clear choice when a license is available. JuMP can also call other SOCP solvers, but IPOPT and Gurobi were the only two that worked straight out of the box.

MULTIPLE SHOOTING

A trajectory is defined as a set of *N* nodes, where each node has position, velocity, mass, and thrust. For each iteration of the SQP algorithm, the relative time between each node is fixed, though the total time of flight is an optimization variable. For each pair of nodes, the state is propagated forward halfway from the first and backward halfway from the second. Continuous thrust is held constant for each node. The defect $\vec{d}$ in state at the midpoint is constrained to be zero.

The Jacobian of the defect constraints with respect to all optimization variables is used to linearize the dynamics constraints. Forward finite differencing is used to approximate each of the partial derivatives in the large matrix. The size of the Jacobian matrix grows with N^2, where N is the

number of nodes used. However, the number of nonzero elements grows with N. Since the Jacobian is sparse, we save computation time by computing only the partials that may be nonzero.

The optimal control problem is cast in terms of an SOCP largely by enforcing a linearization of the dynamics constraints:

$$\vec{d}_k + \left.\frac{\partial \vec{d}}{\partial \vec{X}}\right|_k \cdot \delta\vec{X} = \vec{0} \tag{12}$$

where $\vec{d}_k$ is the vector of all defect constraints at the current iteration. Note that Eqn. (12) is equivalent to the more general Eqn. (10). The cost function of the original problem given in Eqn. (5) is truly quadratic, so that no approximation needs to be made. The objective can be left as-is.

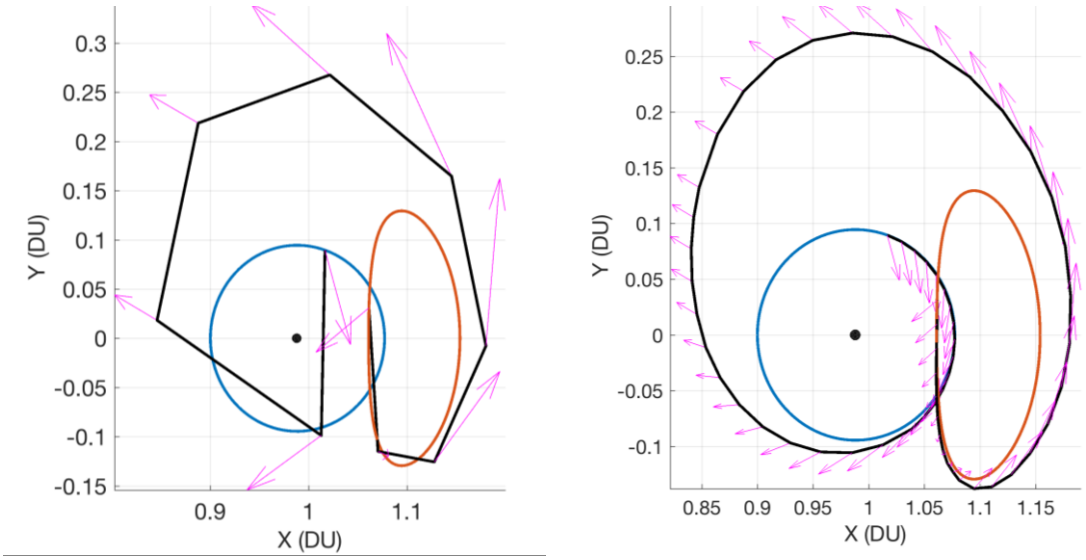
If the endpoints (initial and final nodes) and time of flight are held constant, then the dynamics constraints and objective function alone are enough to find optimal solutions. Adding these few extra variables to the optimization problem significantly increases its complexity, so a full section is devoted to it below.

Fixed-step integration is used, for three reasons. First, when partial derivatives are computed via finite differencing, fixed-step integration yields much more consistent derivatives than adaptive-step integration. Each adaptive-step integration adds a small amount of algorithmic noise because the derivatives function is evaluated at slightly different times for a perturbed initial state. When this noise is divided by a small perturbation size in finite differencing, the noise is amplified and can in some cases be on the order of the derivative itself. Second, the integration is faster because the algorithm does not get hung up propagating near a gravity well. Finally, the amount of work done to propagate each node is identical, which is favorable for future implementation with parallel computing, either on multi-threaded CPU or on GPU.

Mesh Refinement

The reader may question: What about when the trajectory does go near a gravity well? Will the integration accuracy not degrade? To address this important concern, we implement mesh refinement, where nodes are added (or removed) such that the integration error remains within some fixed bounds. Runge-Kutta 78 integration is used where a 7^{th} order polynomial is used to propagate the state, and the 8^{th} order term is used to estimate the truncation error. Typically, an adaptive-step integrator would use the 8^{th} order term to determine each integration step size. Here, we use a fixed step size and instead use the 8^{th} order term to determine where more nodes are needed.

We see in Figure 2 how a coarse initial solution can be refined into a more accurate solution. The mesh refinement operation is fast, requiring on the order of tens of milliseconds. In addition to improving the numerical integration accuracy, mesh refinement tends to assist with the linearization of the problem. Portions of a trajectory which are deep in a gravity well have quickly-changing dynamics, and changes to those nodes have a great effect on the rest of the trajectory. Mesh refinement addresses the accuracy and sensitivity of these nodes simultaneously.



(a) **(b)**

Figure 2. An exaggerated example of the mesh refinement operation. Here, a transfer from a DRO to an L_2 halo orbit was first solved with just 10 nodes in (a), then refined to (b). The plots are in the Earth-Moon rotating frame. Control is held constant for each node.

ENDPOINT CONSTRAINTS

In the simplest terms, we are solving a two-point boundary value problem. It was found that using fully-constrained endpoints (i.e. fixed position, velocity, and time) made the problem much easier to solve because the linear approximation breaks down for the endpoints sooner than for the other optimization variables. However, for many mission design scenarios, we need to be able to optimize the endpoints and time of flight in addition to the intermediary states and controls. When the end states are instead constrained to lie anywhere on an orbit, the 6-state constraint is reduced to a 5-state constraint. If the initial and final orbits are Keplerian, then it is natural to describe the state as a set of orbital elements where, for instance, true anomaly is allowed to drift freely. If the end states are instead constrained to lie on periodic three-body orbits, then there is no equivalent set of only 5 elements to constrain. Instead, we must use a linear or quadratic Taylor series expansion of the three-body orbit with respect to the "true anomaly". We will use the angle τ to be similar to true anomaly and represent the angular distance traveled around a general periodic orbit.

For any arbitrary dynamics and endpoint, we define the function $\vec{q}(\tau)$ to be the actual endpoint state in Cartesian position and velocity as a function of the angle τ. If $\vec{q}$ represents a periodic orbit, then $\vec{q}(0) \equiv \vec{q}(2\pi)$. As implemented in this work, $\vec{q}(\tau)$ is defined by a set of states given at 100 equally-spaced values of τ, read from a text file. The endpoints are interpolated from the set of 100 states given.

The Taylor series expansion of the endpoints are found as follows. First, compute the partial derivatives of the endpoint with respect to τ via finite differencing.

$$\frac{\partial \vec{q}(\tau_k)}{\partial \tau} \approx \frac{\vec{q}(\tau_k + h) - \vec{q}(\tau_k - h)}{2h} \tag{13}$$

$$\frac{\partial^2 \vec{q}(\tau_k)}{\partial \tau^2} \approx \frac{\vec{q}(\tau_k + h) - 2\vec{q}(\tau_k) + \vec{q}(\tau_k - h)}{h^2} \tag{14}$$

Note that with only three evaluations of $\vec{q}(\tau)$, we can estimate both the first and second derivatives. Then, we can form an approximate, quadratic form of the endpoint orbit from the first two terms of the Taylor series.

$$\vec{q}(\tau) \approx \vec{q}(\tau_k) + \frac{\partial \vec{q}(\tau_k)}{\partial \tau} \cdot \delta\tau + \frac{1}{2}\frac{\partial^2 \vec{q}(\tau_k)}{\partial \tau^2} \cdot \delta\tau^2 \tag{15}$$

In SOCP, we can use up to linear equality constraints and up to a quadratic objective, but we cannot use a quadratic equality constraint. It was found that using only a linear equality constraint, the optimization algorithm would tend to bounce around the optimal τ, never quite converging. We developed two solutions to this problem. The first solution is to introduce a quadratic term to the objective function which measures the distance between the quadratic expansion and the linear expansion. Adding this quadratic cost to the path cost yields the following objective function:

$$f = f_{path} + \beta \cdot \left\| \frac{\partial^2 \vec{q}(\tau)}{\partial \tau^2} \right\| \cdot \delta\tau^2 \tag{16}$$

where β is a scaling term. If β is too small, then the solution bounces around the optimal τ indefinitely. If β is too large, then the solution will converge quickly (but prematurely) on a sub-optimal value of τ.

The other approach to help the endpoints converge is simpler: only allow τ to vary on some iterations. For instance, constrain τ to be fixed on the even iterations and allowed to vary on the odd iterations. When the solution is "nearly" converged, then we can safely fix the endpoints and know that we are at a "nearly" optimal solution. As mentioned before, once the endpoints are fixed, the problem tends to converge the rest of the way within a few iterations. Both approaches for optimizing the endpoints are somewhat ad-hoc, and admittedly are not guaranteed to converge to the optimal endpoints. However, it was found that the endpoints do consistently converge to similar values, giving us confidence that the algorithm is effective. It is important to note that for low-thrust propulsion, the endpoint orbits are departed or arrived at gradually, making the exact value of the τ variables less important than for impulsive-burn trajectories. For instance, two solutions could be identical if τ and time of flight are adjusted together – one solution would simply have a coasting period at the end.

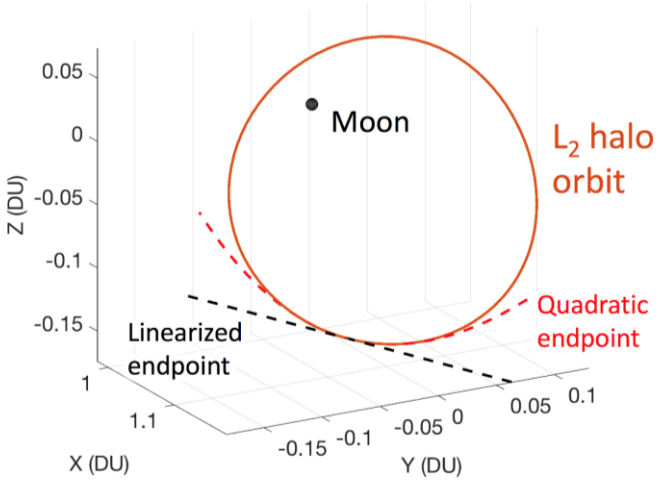


Figure 3. The linear and quadratic approximations of an Earth-Moon L_2 halo orbit. Both the linearized endpoint and the quadratic endpoint are tangent to the halo orbit at one point, but the quadratic endpoint remains close to the halo orbit for a longer time.

Constraining the first and last nodes of the trajectory to lie on a linear expansion of a 3-body orbit was found to be effective. The endpoints are constrained to lie anywhere along a 6-dimensional line (3 dimensions for position, 3 for velocity). Figure 3 shows an example of a linear and quadratic expansion of an L2 halo orbit.

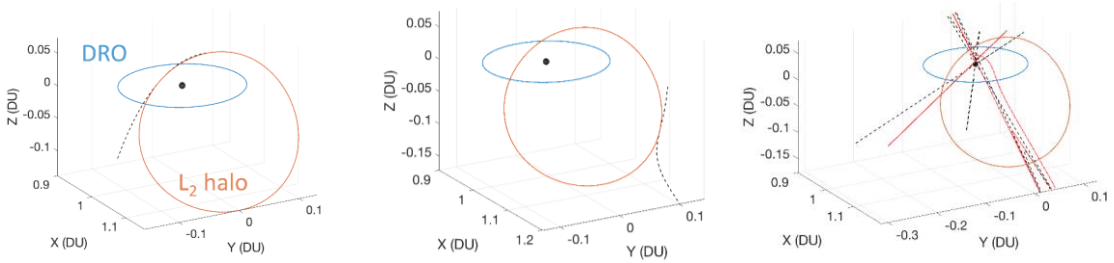


Figure 4. Linear expansions of the modified equinoctial elements representation of a halo orbit. The black dotted line is the linear expansion. The approximation holds well near position (a), less well at position (b), and breaks down at position (c) due to the singularity at 180° inclination.

As mentioned earlier, two-body orbits can easily be parameterized with orbital elements, making it natural to constrain the 5 constant elements and optimize the final element. We explored using Modified Equinoctial Elements relative to the Moon to describe a halo orbit and found that the orbital element set breaks down due to singularities. Hintz compiled a list of 22 orbital element sets[15] and found that each one has at least one singularity. In the case of the Modified Equinoctial Elements, the singularity is at inclination of 180°. For two-body orbits, such a singularity is easy to avoid, but even a simple three-body orbit can cross every possible singularity. Figure 4 shows how, in some cases, a linear expansion of a halo orbit represented with Modified Equinoctial Elements is more accurate than a linear expansion of the same orbit represented with Cartesian position and velocity. In other cases, however, the orbital elements are far less accurate because of the singularity.

EXAMPLES & RESULTS

The greatest benefits from the approach described in this paper are the robustness to poor initial guess and the small number of iterations required to converge on solutions. To demonstrate these qualities, we provide the worst-possible initial guess: random noise. The states guesses are pulled from a random number generator, and the controls guess is zero. Even with such a poor initial guess, the algorithm will still converge quickly.

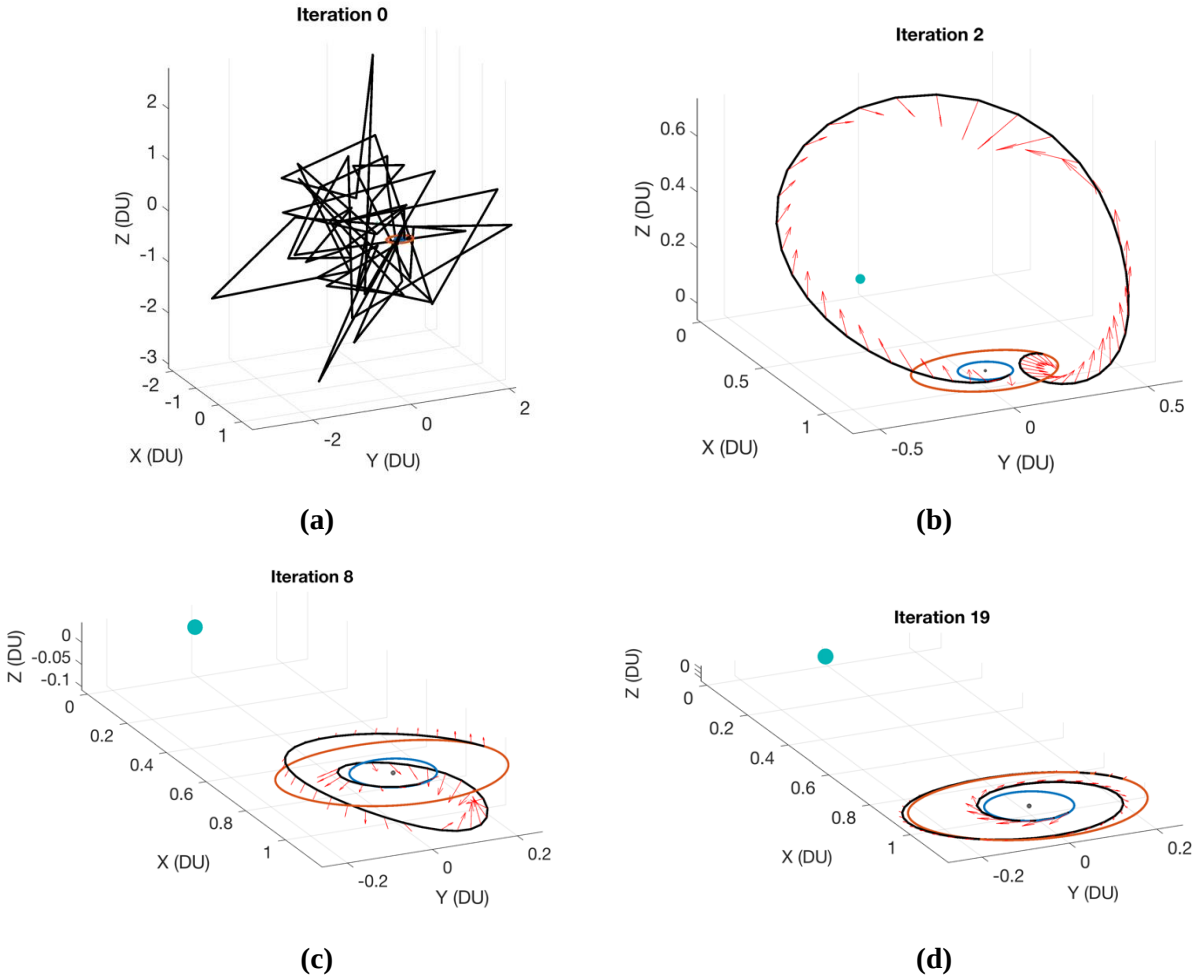


Figure 5. Progression from random noise in (a) to a converged solution in (d) with 19 iterations, for a DRO-to-DRO transfer with fixed endpoints and time of flight. The first DRO (in blue) has an x-crossing at 0.9 DU. The second DRO (in orange) has an x-crossing at 0.8 DU. Earth and Moon are plotted to scale in the synodic reference frame. The canonical distance unit (DU) is used where 1 DU is equal to the average distance between Earth and Moon.

This ability to rapidly converge on an optimal solution is demonstrated in Fig. 5. In almost any situation, it is possible to come up with a better initial guess than random noise. Thus, this result should be considered a worst case. If we really have no knowledge of what the solution should look like, even a random guess is close enough to converge. It should be noted that this approach works well for transfers up to about two complete revolutions. Beyond that, the linearity assumptions in the dynamics constraints break down, and the line search step becomes increasingly necessary. Future work will seek to improve how the algorithm handles multiple-revolution transfers.

We will now show some more example solutions for transfers between DRO's and halo orbits. All of these solutions began with a random initial guess. Thrust was unconstrained in most cases, but the objective function (integral of the control squared) encouraged low-thrust solutions. From prior work, we know that each of these can be converted into a nearby minimum fuel solution with thrust magnitude requirements less than or equal to the maximum thrust stated.

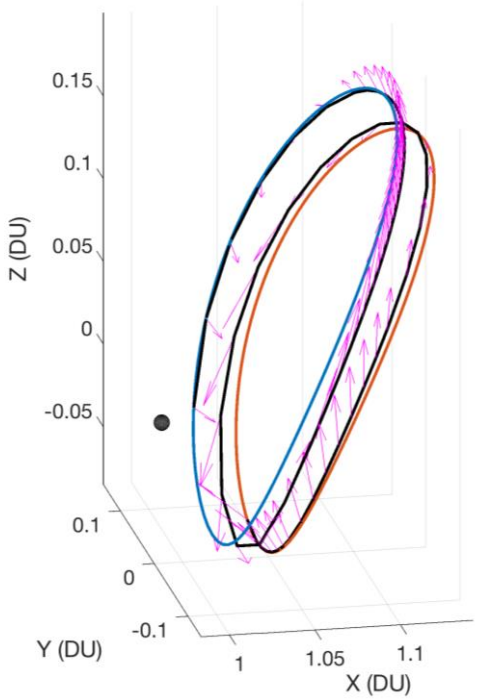


Figure 6. An example transfer from one L_2 halo orbit to another, in the Earth-Moon system. The first halo orbit is in blue and the second in orange. This transfer requires 26 days and an acceleration of 3.5E-5 m/s^2 (equivalently, 35 mN thrust for a 1000 kg spacecraft).

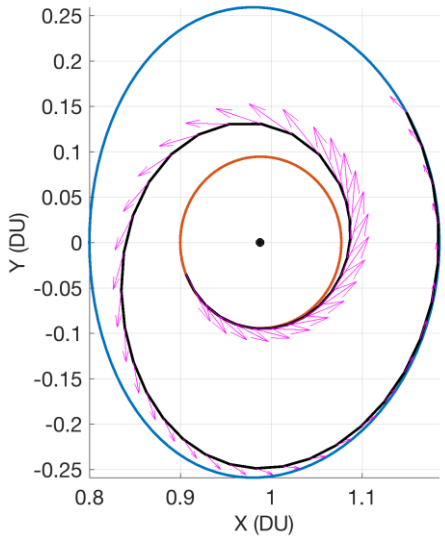


Figure 7. Example transfer from a DRO to another DRO. The first DRO is in blue and crosses the x-axis at 0.8; the second DRO is on orange and cross the x-axis at 0.9. This transfer requires 15 days and an acceleration of 1.7E-4 m/s^2 (equivalently, 170 mN for a 1000 kg spacecraft).

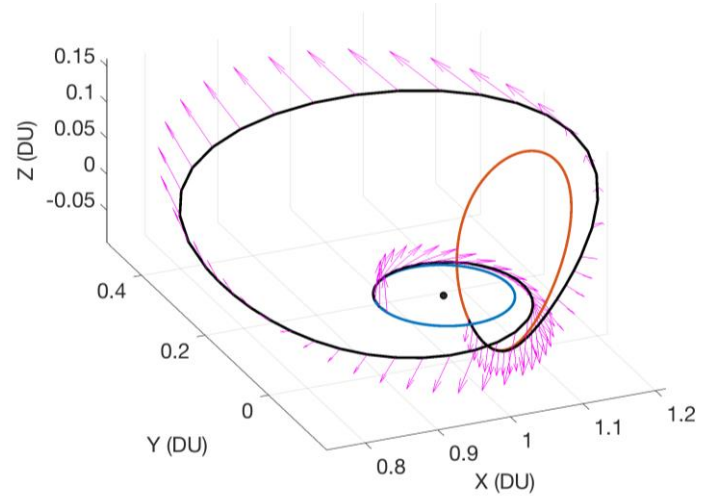


Figure 8. An example transfer from a DRO to an L_2 halo orbit. The DRO is in blue and the halo orbit in orange. This transfer requires 29 days and an acceleration of 2.8E-4 m/s^2 (equivalently, 280 mN for a 1000 kg spacecraft).

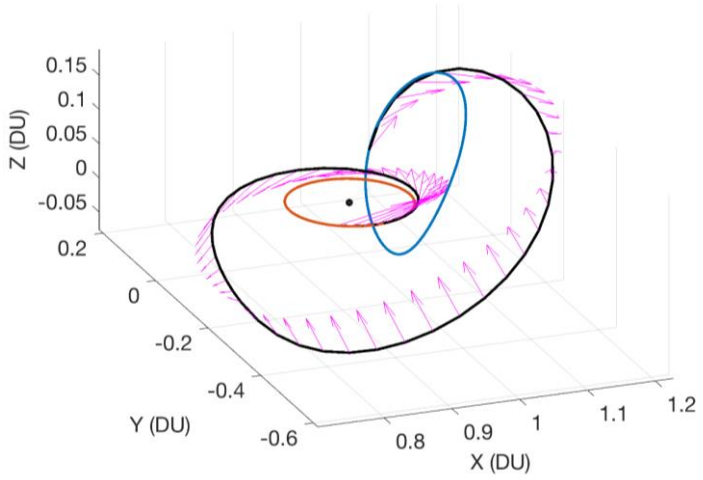


Figure 9. An example transfer from an L_2 halo orbit to a DRO (the reverse of Fig. 7). The halo orbit is in blue and the DRO in orange. This transfer requires 28 days. The max control acceleration was constrained to be 2.8E-4 m/s^2 to match the prior example.

Figures 6-9 provide some example optimal solutions. Note that the thrust levels required are within the range of what current electric propulsion technology can provide. Future work will explore many more optimal solutions between more 3-body orbit types.

Computation Time

Based on the authors' experience with other tools capable of solving similar problems, it was found that the present implementation compares favorably. Future work will include a detailed comparison of this tool with others. For now, we have results for the typical time required by the present research to solve the example problems shown.

Table 1. Comparison of computation time for each part of an iteration

Iteration segment	*Typical time*
Set up SOCP problem	0.2 – 0.5 seconds
Solve SOCP problem	0.2 – 0.5 seconds
Line search	0.2 – 0.5 seconds

As Table 1 shows, the computational load is well balanced across the three main segments of an iteration: setting up the SOCP problem, solving the SOCP problem, and performing the line search. Each requires about 0.2 seconds for 40 nodes or about 0.5 seconds for 100 nodes. The SOCP setup time largely consists of computing the Jacobian of dynamics constraints with respect to states and controls. The Jacobian calculation can potentially be accelerated by computing the Jacobian in parallel on the GPU. Solving the SOCP problem is done by the Gurobi solver, which is already highly optimized and one of the fastest solvers available. We do not expect that any improvement can be made to solving the SOCP problem, and such work is considered outside the scope of the present research. The line search can be accelerated by using a more efficient method to find the best α. By accelerating the SOCP problem setup and the line search, we expect that the total process can be accelerated by a factor of two still.

The exact number of iterations required varies depending on the quality of the initial guess. The number if iterations stated is a typical value from running these transfers several times with different random initial guesses. The total number of iterations required for a problem with fixed endpoints is about 10-20, meaning the total solution time is roughly 2-10 seconds starting from a random initial guess. With a close initial guess, the number of iterations required is typically 2-5, meaning 1-2 seconds to a converged, optimal solution. When the endpoints are optimized, the algorithm typically requires about twice as many iterations.

CONCLUSION

In comparison to the traditional approach of plugging the problem into a "black-box" NLP solver, the methods shown converge even when given no knowledge of the solution at all. It was found that the only piece of information that the user needs to provide is a rough guess for the time of flight, as the transfer time guess will dictate which set of local solutions the algorithm could converge on. This robustness to initial guess is a compelling feature, as three-body orbit transfers are challenging to design with intuition alone. Of course, if a high-quality initial guess *is* available, the methods shown are still valid.

We have shown that endpoints can be efficiently constrained to lie on 3-body repeating orbits, and that time of flight can be optimized as well. When optimizing the endpoints, we must make a trade between converging quickly on sub-optimal endpoints or converging more slowly on endpoints that are arbitrarily close to optimal. It is easy for the mission design engineer to adjust this trade based on the problem at hand.

The biggest limitation to the algorithm at this point is that multi-revolution transfers (greater than 2 revolutions) do not work nearly as well. This restriction comes in because the relationship between node 1 and node *N* becomes increasingly nonlinear as the angular distance grows. Transfers with more than about 1.5 complete revolutions generally require the line search to improve convergence.

Future work includes: Comparison of this algorithm with other established tools; improvements to how multiple-revolution transfers are handled; parallelization of the Jacobian computation; increased efficiency for the line search; and optimization of many more trajectories between a variety of 3-body orbits.

ACKNOWLEDGMENTS

This work was supported by a NASA Space Technology Research Fellowship.

REFERENCES

[1] Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," *SIAM Journal on Optimization*, vol. 12, 2002, pp. 979–1006.

[2] Wachter, A., and Biegler, L. T., "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, 2006, pp. 25–57.

[3] Becerra, V. M., "Solving complex optimal control problems at no cost with PSOPT," *Proceedings of the IEEE International Symposium on Computer-Aided Control System Design*, 2010, pp. 1391–1396.

[4] Parrish, N. L., Parker, J. S., Hughes, S. P., and Heiligers, J., "Low-Thrust Transfers from Distant Retrograde Orbits to L2 Halo Orbits in the Earth-Moon System," *6th International Conference on Astrodynamics Tools and Techniques*, Darmstadt: 2016.

[5] Heiligers, J., Mingotti, G., and McInnes, C., "Optimisation of Solar Sail Interplanetary Heteroclinic Connections," *2nd Conference on Dynamics and Control of Space Systems*, Rome: 2014.

[6] Englander, J., Conway, B., and Williams, T., "Automated Interplanetary Trajectory Planning for Multiple Fly-by Interplanetary Missions," 2012.

[7] Sims, J., Finlayson, P., Rinderle, E., Vavrina, M., and Kowalkowski, T., "Implementation of a Low-Thrust Trajectory Optimization Algorithm for Preliminary Design," *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, 2006, pp. 1–10.

[8] Parrish, N. L., and Scheeres, D. J., "Low-Thrust Trajectory Optimization with No Initial Guess," *26th International Symposium on Space Flight Dynamics*, Matsuyama: 2017.

[9] Bai, X., Turner, J. D., and Junkins, J. L., "A Robust Homotopy Method for Equality Constrained Nonlinear Optimization," 2008.

[10] Bai, X., Turner, J. D., and Junkins, J. L., "Bang-bang Control Design by Combing Pseudospectral Method with a novel Homotopy Algorithm," 2009, pp. 1–14.

[11] Haberkorn, T., Martinon, P., and Gergaud, J., "Low Thrust Minimum-Fuel Orbital Transfer: A Homotopic Approach," *Journal of Guidance, Control, and Dynamics*, vol. 27, 2004, pp. 1046–1060.

[12] Norcedal, J., and Wright, S. J., *Numerical Optimization*, New York: Springer, 2006.

[13] Dunning, I., Huchette, J., and Lubin, M., "JuMP: A Modeling Language for Mathematical Optimization," *SIAM Review*, vol. 59, 2015, pp. 1–25.

[14] Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B., "Julia: A fresh approach to numerical computing," *arXiv*, vol. 59, 2015, pp. 1–37.

[15] Hintz, G. R., "Survey of Orbit Element Sets," *Journal of Guidance, Control, and Dynamics*, vol. 31, 2008, pp. 785–790.